Simple Experiments for UG Using Arduino

Sandeep S Ghugre

UGC DAE CSR KC
Email : ssg.iuc@gmail.com
Website : www.iuc.res.in

May 5, 2021

Colleagues at UGC DAE CSR KC

Figure: Shri Mukesh Kumar

Preamble

UGC-DAE Consortium for Scientific Research, Kolkata Centre, is in the process of developing a range of innovative, low cost experiments based on the routinely available resources for the undergraduates. These experiments are expected to be illustrative and contribute in their understanding of the basics of the subject, apart from rejuvenating the fun factor in the learning process.

These experiments utilize the

1. Sound card in the PC
2. Open source micro-controller : ARDUINO

as the Data Acquisition System.

What is a Microcontroller

A microcontroller (self contained system with processor, memory & peripherals combined on a single hardware real estate), which is an open-source physical computing platform based on a simple i/o board and a development environment

Innovative micro-controller based experiments can be developed which would allow for open-ended experiments in the conventional laboratory.

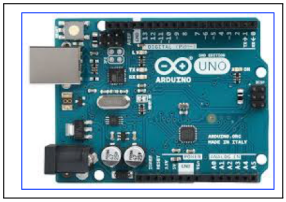
Why ARDUINO

The choice of *Arduino* was essentially due to

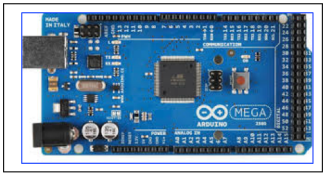
1. It being an *Open Source*, both in terms of hardware and software.
2. The hardware is cheap and can also be built from the components using the information available in public domain.
3. It can be powered from either a USB or a standalone DC power
4. It can run standalone from a computer (chip is programmable) and it has a decent memory.
5. It can communicate with the computer via
   5.1 Serial , Bluetooth , Ethernet

1. It can work with both Digital and Analog signals, Sensors and Actuators
2. Various shields, *ie.* boards that can be plugged on top on the Arduino or it's variants, that enhance it's capabilities are easily available.
3. Ready made boards such as IR transmitter and receiver, Ultra sonic distance sensor, to name a few are available which can be easily integrated with the Arduino microcontroller, allowing us the flexibility to configure experiments as per user requirement and specifications.

Variants of ARDUINO



UNO



Mega

And at the heart of the Arduino development board is the *Atmel ARV Atmega 328* , a modified Harvard architecture 8-bit RISC single chip microcontroller which was developed by Atmel in 1996.

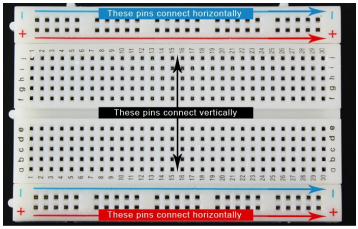
Accessories

To effectively use the Arduino, we require the following

1. USB printer cable : Essential
2. Breadboards
   2.1 **Mini Breadboard**
   2.2 **Small Breadboard**
3. Breadboard Power supply, this has to be used with a **small breadboard**.
4. Connectors : Essential
5. Breakout Boards
6. Components : : As per the requirement of expt.

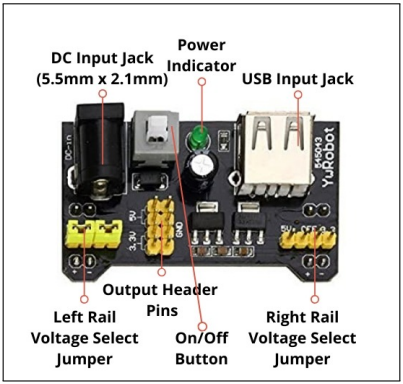
Breadboards

Mini Breadboard

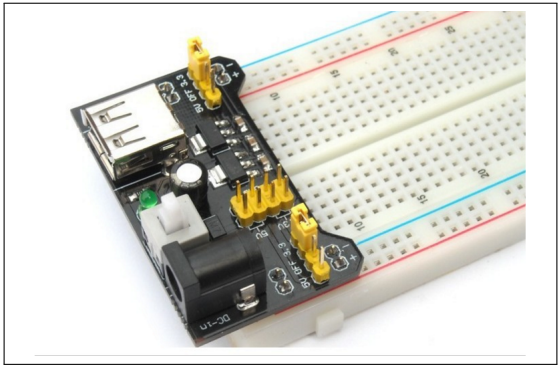


Small Breadboard

Breadboard Power Supply



Breadboard Power Supply



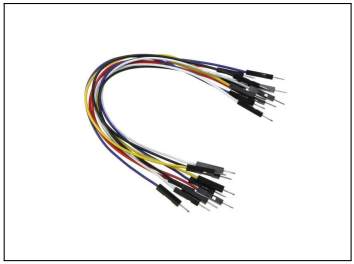
USB Connectors

USB connector for Uno

USB Connector for Nano

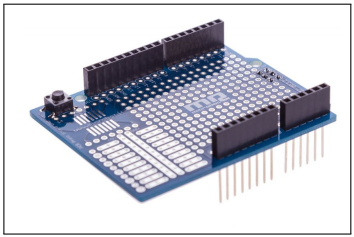
Connectors

9V snap Connector

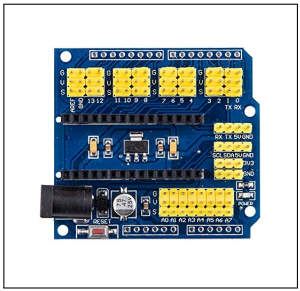


Male-to-Male Connector

Prototype Shields , Breakout Boards



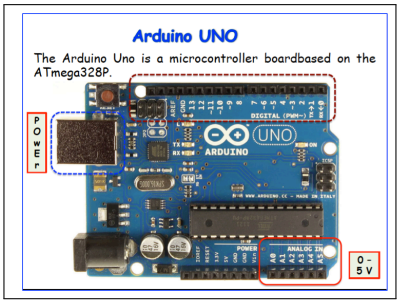
Prototype Shield



Prototype Shield for Nano

Uno

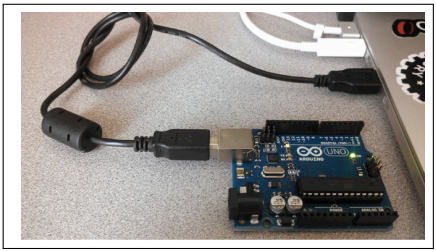
The Arduino UNO is based on the ATmega328P chip.



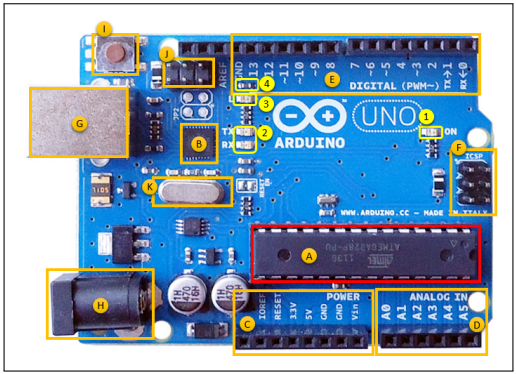
The board has a power LED and also a RESET button. The board can be collected to a host computer using a USB cable.

Connection to the computer

Now all we need to do is connect the microcontroller to the system via a USB and we are all set.



Hardware Real Estate



CPU

Heart of the Arduino development board is the *Atmel ARV Atmega 328P (A)* , 8-bit RISC single chip microcontroller which was developed by Atmel in 1996. It can be identified as a **prominent black rectangular chip with 28 pins**. The components of relevance to us are

1. 32 KB Flash memory.
2. 2KB of RAM.
3. CPU.
4. A non volatile Electrically Erasable Programmable Read Only Memory (EEPROM).

I/O Pins :Analog

The Arduino Uno has 6 analog pins, which utilize a **10 bit** ADC (Analog to Digital converter).

They can accept a **maximum** voltage of 5 volt.

They are labelled as Pins A0-A5.

These pins just measure voltage and not the current because they have very high internal resistance. Hence, only a small amount of current flows through these pins.

Pins labelled 0-13 of the Arduino Uno serve as digital input / output pins.

Pin 13 of the Arduino Uno is connected to the built-in LED.

When digital pins are used as output pins, it is recommended to limit the current to 20 milliamps.

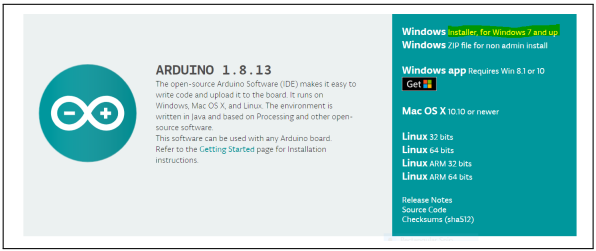
In the Arduino Uno - pins 3,5,6,9,10,11 labelled by $\sim$ have Pule Width Modulation capability which simulate an analog like output.

Misc.

1. Crystal Oscillator This is a quartz crystal oscillator which ticks 16 million times a second. On each tick, the microcontroller performs one operation.
2. USB Connector This is a printer USB port used to load a program from the Arduino IDE onto the Arduino board. The board can also be powered through this port.
3. Power Port The Arduino board can be powered through an AC-to-DC adapter or a battery using a conventional **2.1mm center-positive plug** into the power jack of the board.

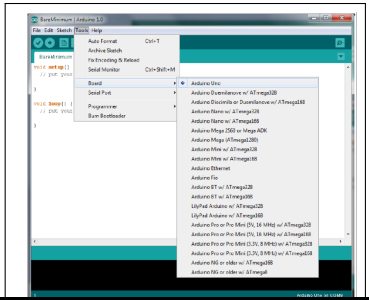
Integrated Development Environment for Arduino

The IDE (Integrated Development Environment) can be downloaded from *http://arduino.cc/en/Main/Software* .

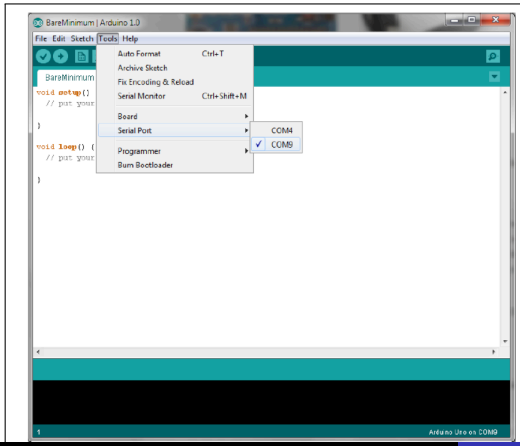


Configure the IDE

We need to configure the development environment, by providing, the board details as well as the port to which it is connected. Most of the time the software will prompt you the correct options :



COM port Selection



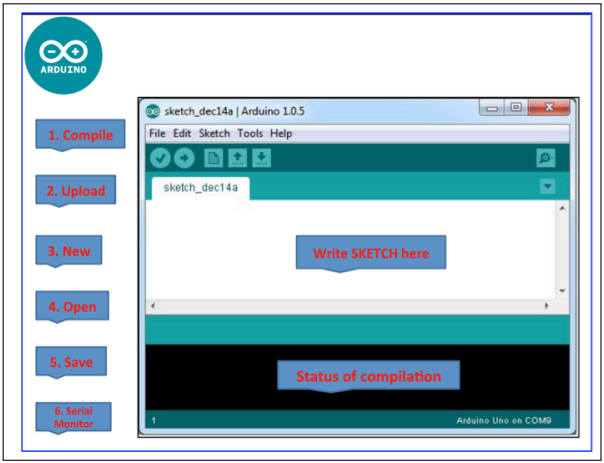
Anatomy of a sketch

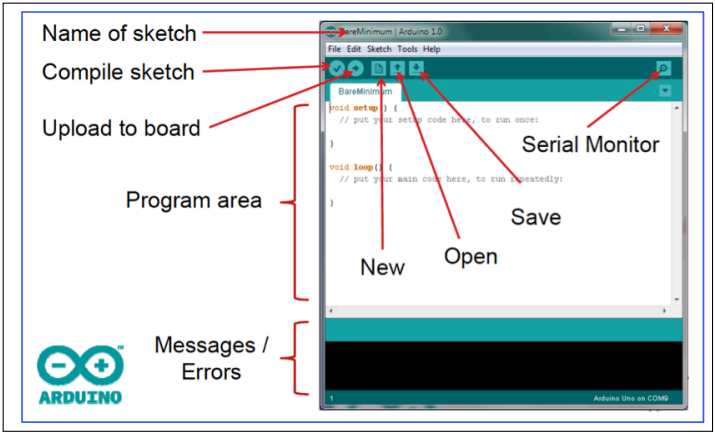
Having done this we are all set to write our first program for the micro-controller which is the equivalent of **Hello World**.

A program in Arduino is referred to as *Sketch*.

The anatomy of a sketch is

void setup() → initialization
executed only when the program begins
void loop() → code executed continously


ARDUINO
1. Compile
2. Upload
3. New
4. Open
5. Save
6. Serial Monitor
sketch_dec14a | Arduino 1.0.5
File Edit Sketch Tools Help
sketch_dec14a
Write SKETCH here
Status of compilation
1
Arduino Uno on COM9


Name of sketch
Compile sketch
Upload to board
Program area
Messages / Errors
Serial Monitor
Save
New
Open
BareMinimum | Arduino 1.0
File Edit Sketch Tools Help
BareMinimum
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
Arduino Uno on COM9
ARDUINO

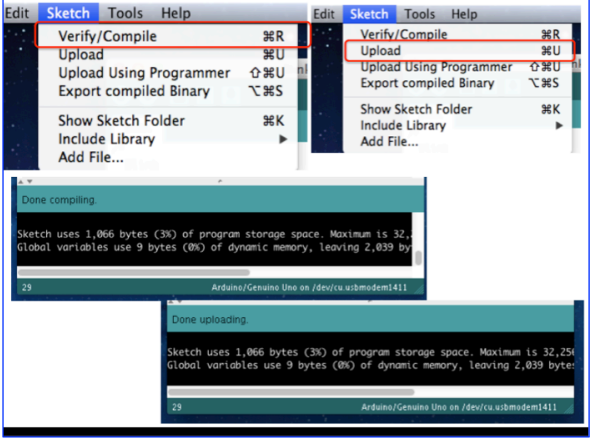
To test if we have set it up correctly, we could try to blink the on-board LED, which is connected to pin 13

```
LED_BUILTIN = 13;

void setup() { pinMode(LED_BUILTIN, OUTPUT); }

void loop() { digitalWrite(LED_BUILTIN, HIGH);

delay(1000);

digitalWrite(LED_BUILTIN,LOW);

delay(1000); }
```

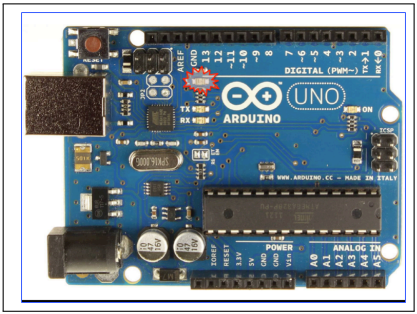
Uploading the code to the computer

The procedure to transfer this code onto the Arduino is as follows :

1. The sketch, which is case sensitive, is typed in the program area.
2. Press the Compile button, for compilation and any errors are identified in this process and need to be rectified before the code is compiled.
3. Press the *Upload* button, to program the Arduino board with the sketch.
4. During the uploading, the *TX/RX* LED will flash, indicating a communication between the Arduino and the PC.



Once the code gets uploaded on to micro-controller, the onboard LED which is connected to *pin 13* will blink. The blinking rate can be controlled by modifying the option parameter for *delay*. The parameter is set in milli-seconds



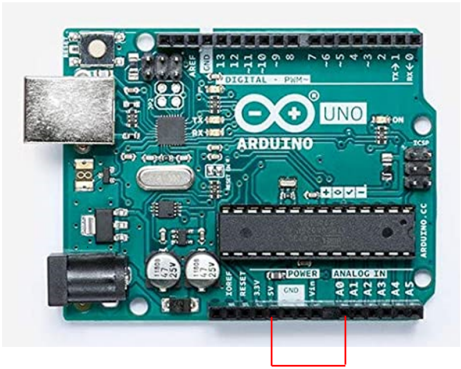
ADC on the ARDUINO

An **A**nalog-to- **D**igital **C**onvertor, converts the analog voltage which is provided as input into an equivalent digital number.

We classify (specify) the given ADC based upon it's bits.

For example if we have a *10* bit ADC, then it will map V_{min} to $V_{max} \rightarrow 2^{10} = 1024$ digital numbers.

Arduino accepts analog signals upto 5 volt and hence has a resolution of about 5 milli-Volt.



Measure Analog Voltage Using Arduino

analogRead(pin_number)

We know that, the ADC after converting the input analog voltage would return an integer value (channel number).

The inbuilt function *analogRead(pin_number)*, would help us obtain the ADC value for the given input analog voltage.

```
int SensorValue = analogRead(A0);
```

The above command, would store the ADC channel number (as an integer) corresponding to the analog voltage connected to Analog Pin Number 0.

SerialMonitor

Now, the value obtained would have to be displayed to the user.

A simple solution would be to use the Serial Monitor for this purpose.

```
Serial.begin(9600)
```

We need to configure the BAUD rate in the initialization routine Setup, which in the present case is set to 9600 bits per second.

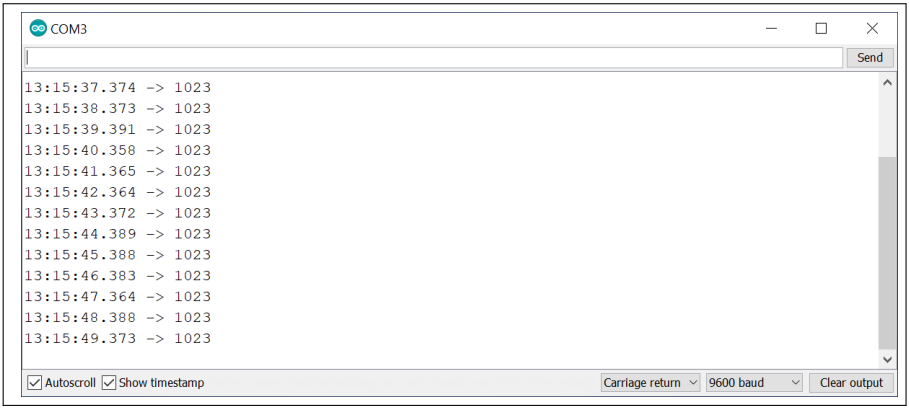
Read Analog Voltage using IDE : Raw ADC value

```
1  // the setup routine runs once when you press reset://'
2  void setup() {
3  // initialize serial communication at 9600 bits per second:
4    Serial.begin(9600);
5  }
6
7  // the loop routine runs over and over again forever:
8  void loop() {
9  // read the input on analog pin 0:
10   int sensorValue = analogRead(A0);
11 // print out the value you read:
12   Serial.println(sensorValue);
13 // delay in the milli-seconds in between reads for stability
14   delay(1000);         // delay in between reads for stability
15 }
```

Once the program is compiled and uploaded you can view the results in the Serial Monitor

Once the program is compiled and uploaded you can view the results in the Serial Monitor, the commands of relevance are shown below.

Tools ⟶ Serial Monitor
⟶ Autoscroll (Off)
⟶ Show timestamp (Off)
⟶ Carriage return
⟶ 9600 baud


COM3
Send
13:15:37.374 -> 1023
13:15:38.373 -> 1023
13:15:39.391 -> 1023
13:15:40.358 -> 1023
13:15:41.365 -> 1023
13:15:42.364 -> 1023
13:15:43.372 -> 1023
13:15:44.389 -> 1023
13:15:45.388 -> 1023
13:15:46.383 -> 1023
13:15:47.364 -> 1023
13:15:48.388 -> 1023
13:15:49.373 -> 1023
Autoscroll
Show timestamp
Carriage return
9600 baud
Clear output

We know that the ADC converts the input into a channel number, and we then need to map it or convert it to the quantity of our relevance. This is referred to as Calibration.
For example, using the **on-board 10 bit ADC**, we have the following relation

$$0\ V \longrightarrow 0 \text{ channel}$$
$$5\ V \longrightarrow 1023 \text{ channel}$$
$$\frac{volt}{channel} = \frac{5}{1023}$$
$$\text{unknown voltage} = \frac{5}{1023} \times \text{channel}$$

Offcourse we have to bear in mind, that while the ADC channel is an integer, the mapped voltage would be a real number.

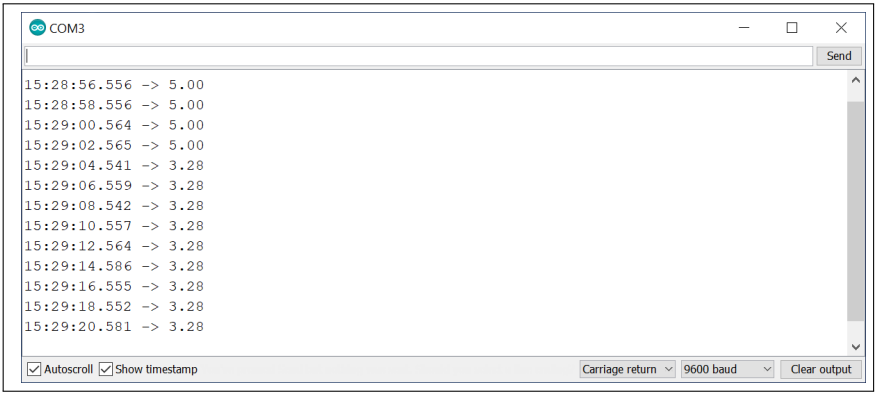
Read Analog Voltage using IDE : Actual Voltage

The ADC Channel Numbers are Integer while the actual Voltages are Real numbers

```
// the setup routine runs once when you press reset:
void setup() {
// initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
// read the input on analog pin 0:
int sensorValue = analogRead(A0);
// Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
float voltage = sensorValue * (5.0 / 1023.0);
// print out the value you read:
Serial.println(voltage);
}
```

Once the program is compiled and uploaded you can view the results in the

We have used the onboard 5 and 3.3 volt supply as the analog signal.



Time Stamped Data

Time stamped data, *ie*, we need to record the time along with the data.

We could use the inbuilt millis() function, which records the number of milliseconds that have elapsed since the program started running.

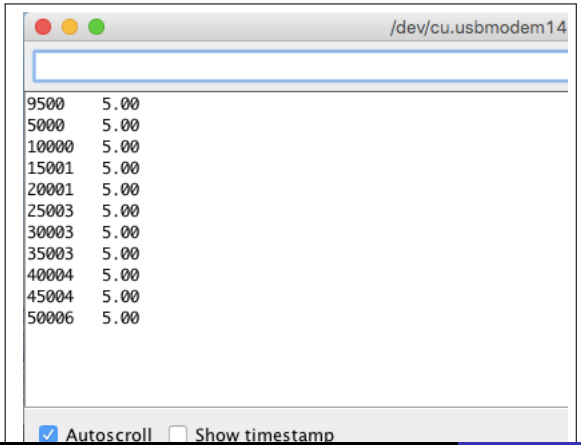
The value returned is an unsigned long.

```
start_tim = millis()                    in the initialization routine
current_tim = millis()                  in the main loop
elapsed_tim = current_tim - start_tim   in the main loop
```

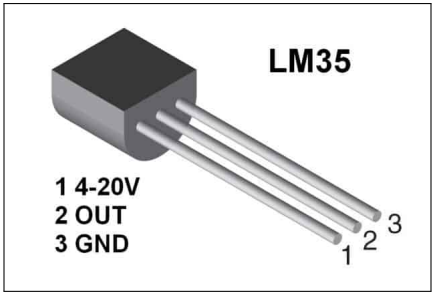
Time Stamped Data

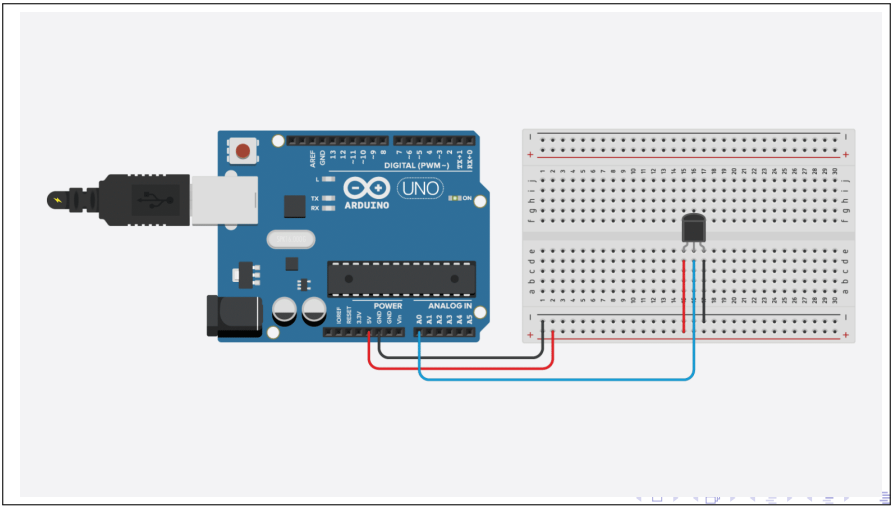
```
unsigned long startMillis;
unsigned long currentMillis;
unsigned long now;
void setup() {
// initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
// initialize start time :
  startMillis = millis();
}

// the loop routine runs over and over again forever:
void loop() {
  currentMillis = millis();
  now = currentMillis - startMillis;
  int sensorValue = analogRead(A0);
  float voltage = sensorValue * (5.0 / 1023.0);
  Serial.print(now);
  Serial.print("\t");
  Serial.println(voltage);
  delay(50);
}
```

Time Stamped Data



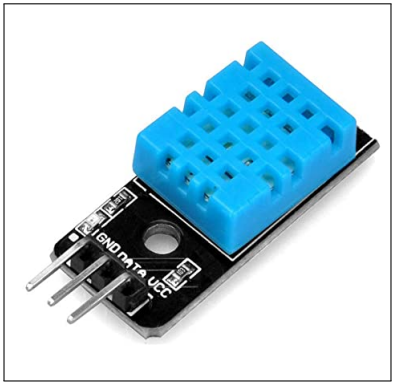
LM 35





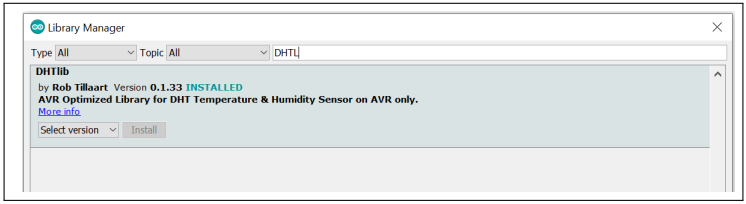
```
/* LM35 analog temperature sensor with Arduino example code. More info: https://www.
    makerguides.com */

// Define to which pin of the Arduino the output of the LM35 is connected:
#define sensorPin A0

void setup() {
  // Begin serial communication at a baud rate of 9600:
  Serial.begin(9600);
}

void loop() {
  // Get a reading from the temperature sensor:
  int reading = analogRead(sensorPin);

  // Convert the reading into voltage:
  float voltage = reading * (5000 / 1023.0);

  // Convert the voltage into the temperature in degree Celsius:
  float temperature = voltage / 10;

  // Print the temperature in the Serial Monitor:
  Serial.print("TEMPRATURE = ");
  Serial.print(temperature);
  Serial.print(" \xC2\xB0"); // shows degree symbol
  Serial.println("C");
```

DHT 11 Module

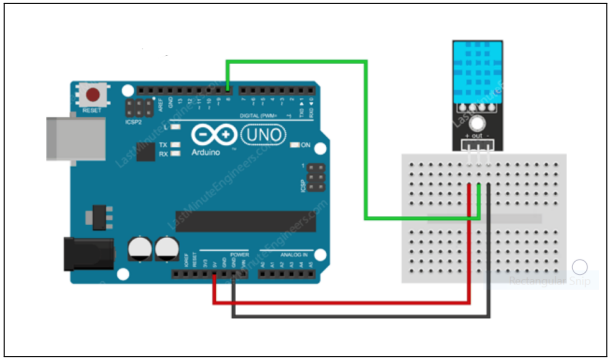


Installing DTHLib

The procedure to add a library is to use the inbuilt library manager Tools $\longrightarrow$ Manage Libraries, type DHTLib, and if the library is not installed it would prompt you to install the same.



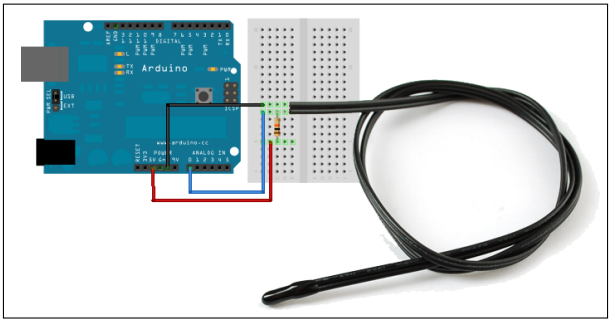
Circuit diagram for DHT 11 Module



Sketch for DHT11

```
#include <dht.h>

dht DHT;

#define DHT11_PIN 8

void setup(){
  Serial.begin(9600);
}

void loop(){
  int chk = DHT.read11(DHT11_PIN);
  Serial.print("Temperature = ");
  Serial.println(DHT.temperature);
  Serial.print("Humidity = ");
  Serial.println(DHT.humidity);
  delay(1000);
}
```

Thermistor and Arduino



Steinhart-Hart equation, attempts to model the **thermistor resistance** as a function of **temperature**, and is given by

$$\frac{1}{T} = A + Bln(R) + C\left[ln(R)\right]^3$$

T : temperature (in Kelvin)

R : resistance at T (in ohms)

$A,\ B$ & C : are the Steinhart Hart coefficients

This equation at times for the NTC thermistors also be characterised with the B or β parameter equation given by

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} ln\left[\frac{R}{R_0}\right]$$

T_0 : room temperature (in Kelvin) $= 25^0\ C = 298.15K$

R_0 : resistance at room temperature $\sim 10K\Omega$

B : in this case $\sim$ 3950coefficient of the thermistor

```
byte NTCPin = A0;
#define SERIESRESISTOR 10000
#define NOMINAL_RESISTANCE 10000
#define NOMINAL_TEMPERATURE 25
#define BCOEFFICIENT 3950

void setup()
{
Serial.begin(9600);
}
void loop()
{
float ADCvalue;
float Resistance;
ADCvalue = analogRead(NTCPin);
Serial.print("Analoge ");
Serial.print(ADCvalue);
Serial.print(" = ");
//convert value to resistance
Resistance = (1023 / ADCvalue) - 1;
Resistance = SERIESRESISTOR / Resistance;
Serial.print(Resistance);
Serial.println(" Ohm");
```

```
float steinhart;
steinhart = Resistance / NOMINAL_RESISTANCE; // (R/Ro)
steinhart = log(steinhart); // ln(R/Ro)
steinhart /= BCOEFFICIENT; // 1/B * ln(R/Ro)
steinhart += 1.0 / (NOMINAL_TEMPERATURE + 273.15); // + (1/To)
steinhart = 1.0 / steinhart; // Invert
steinhart -= 273.15; // convert to C

Serial.print("Temperature ");
Serial.print(steinhart);
Serial.print(" \xC2\xB0");
Serial.println("C");
delay(10000);
}
```

Circuit : Charging of a capacitor

$$Q = C \cdot V_0 \left(1 - e^{\left[\frac{t}{RC}\right]}\right)$$

$$V(t) = V_0 \left(1 - e^{\left[\frac{t}{RC}\right]}\right)$$

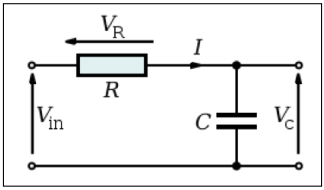


Figure: RC circuit.

Arduino : Charging of a capacitor

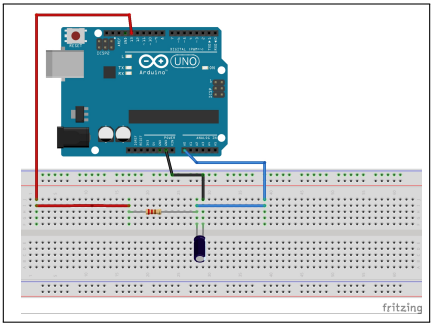


Figure: Charging of a capacitor using Arduino.

Arduino : Charging of a capacitor

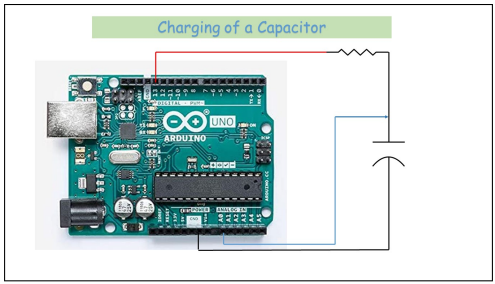


Figure: Charging of a capacitor using Arduino.

Sketch : Charging of a Capacitor

```
int SensorValue = 0 ;
int led = 13 ;
float Voltage1 = 0.0;
long unsigned now = 0.0 ;
long unsigned then = 0.0 ;
long unsigned elapsed = 0.0 ;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600);
  digitalWrite(led, LOW);
  delay(9000);
  digitalWrite(led, HIGH);
  then = millis();
```

```
}

void loop() {
SensorValue = analogRead(A0);
now = millis();
Voltage1 = SensorValue * ( 5.0/1023.0);
elapsed = now - then ;
//Serial.print(elapsed) ;
//Serial.print("\t");
Serial.println(Voltage1);
}
```

Procedure : Charging of a Capacitor

We wait sufficiently long enough (Line 11) for us to fire up the serial terminal, prior to commencing the charging.

We start the charging, by setting the Pin 13 to High State, (Line 12).

On the Serial Monitor we observe the time and the voltage across the capacitor.

These values are seperated by a tab (Line 22).

Accordingly in the analysis routine, we will have to set the delimiter .

Save the data in an ASCII file

The values from the Serial Monitor, are to be copied to an ASCII file, which is achieved, by

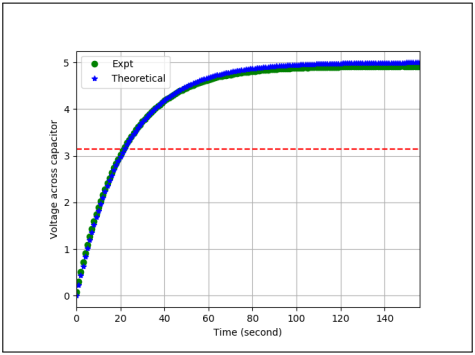
1. Use the command CNTR+A, which *selects all*
2. Copy the selected contents using the familiar CNTR+C command.
3. Open the Python GUI
4. Select the File option and the New File sub-menu
5. In the poped up empty window CNTR+V to copy the contents
6. Save the file as a **.txt** file by choosing the option as *txt* in the *Save As* sub-menu.
7. Scroll down to the end of the file and ensure that the last line is complete. Delete any half written line.

Analysis : Charging of a Capacitor

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import os
4
5  r = 100000
6  c = 220E-06
7  Vin=5.0
8  f1 = open('ssg1.txt','r')
9  header1=f1.readline()       # skip the header
10 data1 = np.genfromtxt(f1,delimiter='\t')
11 #data1 = np.genfromtxt(f1,delimiter=',')
12 y_vr=[]          # array to store the voltage across the resistor
13 y_vc_theo=[ ]   # arry to store the calculated voltage across capacitor
14 temp1 = 0.0     # temporary variable
15 temp2 = 0.0     # temporary variable
16 tau = r*c       # tau of the circuit
17
18 x_time=data1[:,0]
19 y_vc=data1[:,1]
```

```
for i in range(len(x_time)):
        temp1 = 5.0 - y_vc[i]
        y_vr.append(temp1)
        temp2 = Vin-Vin*(np.exp((-x_time[i])/(r*c)))
        y_vc_theo.append(temp2)
        temp1=0.0
        temp2=0.0

f1.close()
X=[0,tau,10*tau,100*tau]
Y=[0.63*Vin,0.63*Vin,0.63*Vin,0.63*Vin] # for observing tau
plt.plot(X,Y,"r--")
plt.plot(x_time,y_vc,"go",label="Expt")
plt.plot(x_time,y_vc_theo,"b*",label="Theoretical")
plt.legend(loc="best")
plt.xlabel('Time (second)')
plt.ylabel ('Voltage across capacitor')
plt.xlim(0,max(x_time))
plt.grid()
plt.show()
```

Plots : Charging of a capacitor



Proximity Sensor

The IR Proximity sensor is based on the principle of IR reflectance.

The IR transmitter emits Infra-Red radiations ($\lambda \sim 700\ nm - 100\ nm$).

Infrared receivers detect the radiation from an IR transmitter, are photo-diodes and they detect only infrared radiation.

This signal is processed by an Op-Amp operated as a Comparator. When, the IR signal is interrupted due to the presence of an obstacle in the line of sight, the comparator, produces a change in the output, say for example a **HIGH** signal.

Commercial Proximity Sensors

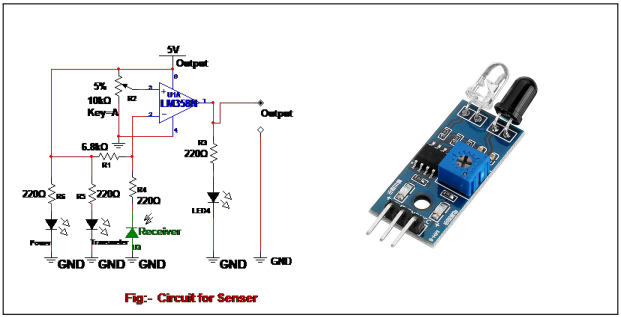


Figure: Proximity Sensor.

Simple Proximity Receiver

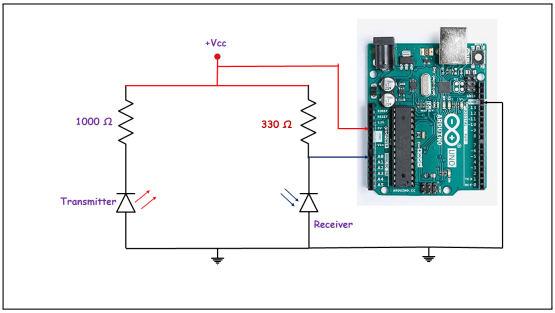
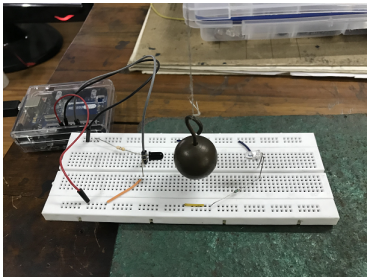


Figure: **IR Transmitter & Receiver** coupled to Arduino.

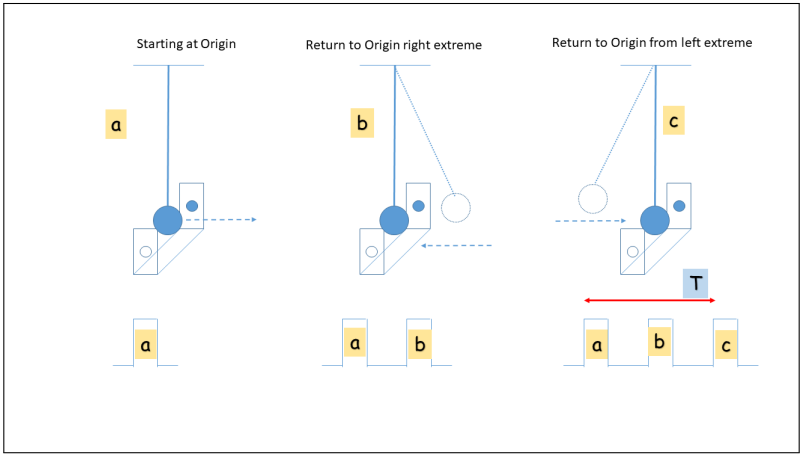
Experimental Setup

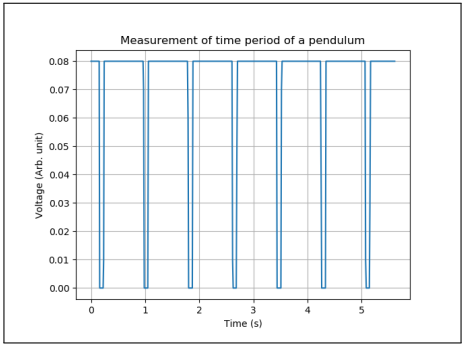


The Sketch is identical to the one used for the charging of the capacitor.

Analysis

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  data = np.genfromtxt("pendulum_data.txt")
5
6  time1 = data[:,0]/1000.0
7  voltage = data[:,1]
8
9  plt.plot(time1,voltage)
10 plt.xlabel('Time (s)')
11 plt.ylabel ('Voltage (Arb. unit)')
12 plt.title ('Measurement of time period of a pendulum')
13 plt.grid()
14 plt.show()
```





Time interval between three *blips* or change in level, would give us the **Time period** for the given pendulum.

Voltage Divider

The configuration comprises of two series resistors R_1 & R_2 across which we apply the **input** voltage, V_i.

$$V_o = V_i \cdot \frac{R_2}{R_1 + R_2}$$

if $R_1 = R_2 = R$ then

$$V_o = V_i \cdot \frac{R}{R + R} = \frac{V_i}{2}$$

if $R_2 >> R_1$ then

$$V_o \sim V_i \cdot \frac{R_2}{R_2}$$
$$\sim V_i$$

if $R_2 << R_1$ then

$$V_o \sim V_i \cdot \frac{0}{R_1} \sim 0$$

Light Dependent Resistor

A **Light Dependent Resistor** is a component that is sensitive to light.

LDRs are passive components, made from semiconductor materials to enable them to have their light sensitive properties.

Interaction of light photons with the semiconductor lattice results in the transfer of the photon energy, to the electrons.

These energetic electrons break free from crystal lattice and are available for conduction.

Arduino and LDR

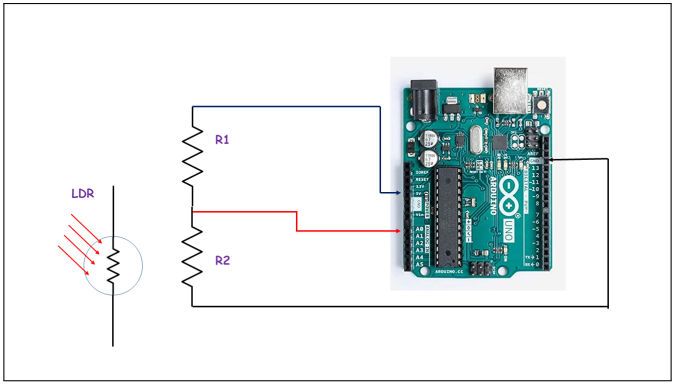
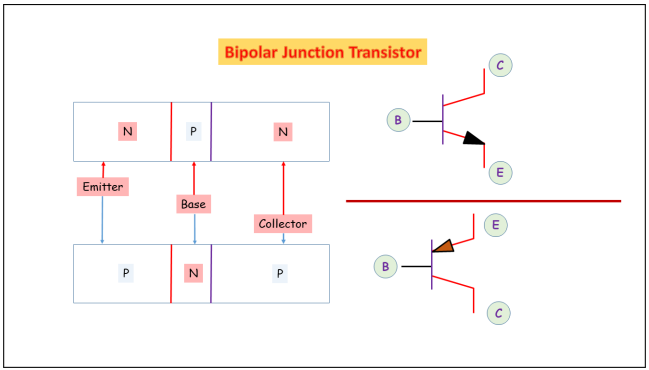


Figure: LDR connected in a voltage divider (replacing R_2) and coupled to an

Measure Resistance of LDR

```
1
2  // the setup routine runs once when you press reset:
3  void setup() {
4    // initialize serial communication at 9600 bits per second:
5    Serial.begin(9600);
6  }
7
8  // the loop routine runs over and over again forever:
9  void loop() {
10   // read the input on analog pin 0:
11   int sensorValue = analogRead(A0);
12   // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
13   float voltage = sensorValue * (5.0 / 1023.0);
14   float vratio = voltage/5.0;
15   float Rx = -1000.0/(vratio-1);
16   // print out the value you read:
17   Serial.print(voltage);
18   Serial.print("\t");
19   Serial.println(Rx);
20   delay(5000);
21 }
```

Bipolar Junction Transistor



BJT as a Switch

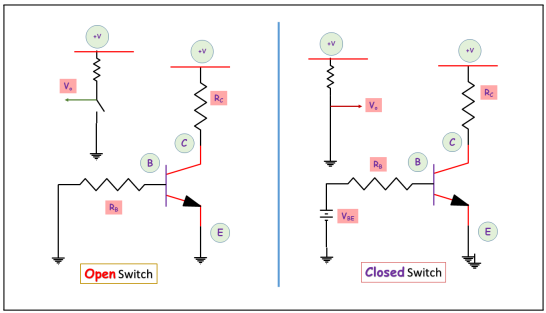


Figure: BJT as a **switch**.

Arduino : Transistor as a switch

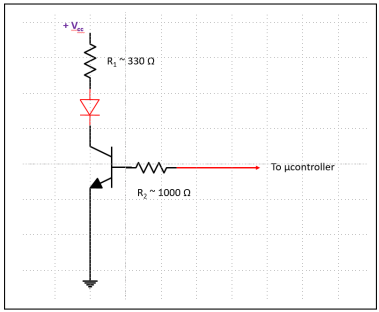


Figure: Transistor as a switch.

Determination of Planck's Constant

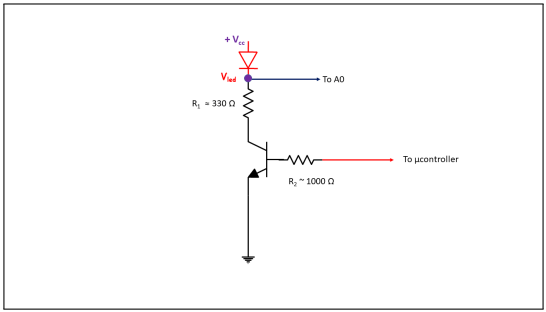


Figure: Determination of h.

Determination of h

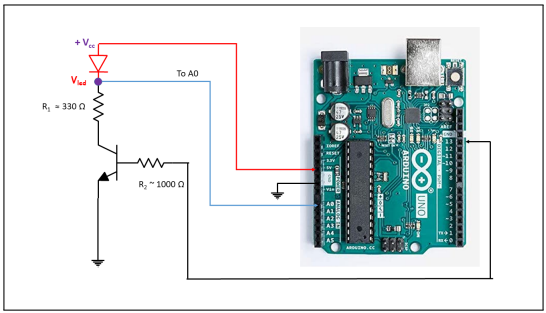


Figure: Transistor as a switch.

Determination of h.

```
int ledPin = 13;     // LED connected to digital pin 13
int SensorValue =0;
float Voltage =0.0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600);
}

void loop() {
    digitalWrite(ledPin, HIGH);
    SensorValue=analogRead(A0);
    Voltage = SensorValue *(5.0/1023.0);
    Serial.println(Voltage);
   delay(5000);
   digitalWrite(ledPin, LOW);
    SensorValue=analogRead(A0);
    Voltage = SensorValue *(5.0/1023.0);
    Serial.println(Voltage);
   delay(5000);
}
```

Now, we know that the v_g, is related to the Planck's Constant h as

$$h = \frac{e \times V_g \times \lambda_g}{c}$$

Color	$\lambda_g\|_{nm}$	$V_g\|_{volt}$	$h\|_{j-s}$
Red	665	1.84	6.36×10^{-34}
Yellow	590	1.95	6.52×10^{-34}
Green	560	1.94	6.13×10^{-34}
Orange	635	1.88	5.79×10^{-34}

Multivibrator

Multivibrator is an electronic switching circuit, which usually has **two states**.

They are generally classified as

1. Mono-stable — One stable state
2. Bi-stable — Two stable states
3. A-stable
   - 3.1 No stable state
   - 3.2 Keeps on oscillating between the two states

555 Timer

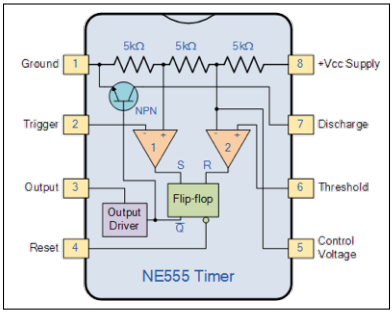


Figure: A simplified *block diagram* of the 555 timer.

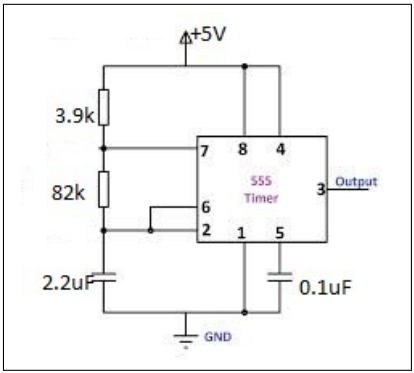


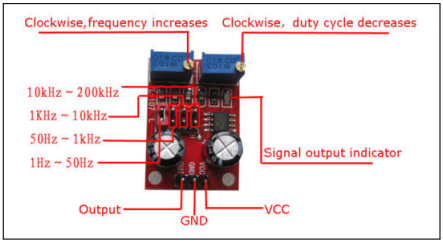
Figure: 555 timer as Astable Multivibrator.

$$\text{Frequency} = \frac{1}{[T_{high} + T_{low}]} = \frac{1.44}{[R_1 + 2 \times R_2] \times C}$$

The *Duty Cycle*, is given by

$$\text{Dugty Cycle} = \left[\frac{T_{high}}{(T_{high} + T_{low})}\right]$$

Arduino Compatible 555 timer



```
1  int SensorValue = 0;
2  float Voltage1 = 0;
3  unsigned long startMillis;
4  unsigned long currentMillis;
5  unsigned long now;
6  void setup() {
7  // initialize serial communication at 9600 bits per second:
8    Serial.begin(9600);
9  // initialize start time :
10   startMillis = millis();
11 }
12
13 // the loop routine runs over and over again forever:
14 void loop() {
15   currentMillis = millis();
16   now = currentMillis - startMillis;
17   SensorValue = analogRead(A0);
18   Voltage1 = SensorValue * (5.0 / 1023.0);
19   Serial.print(now);
20   Serial.print("\t");
21   Serial.println(Voltage1);
22   delay(10);
23 }
```

Waveform

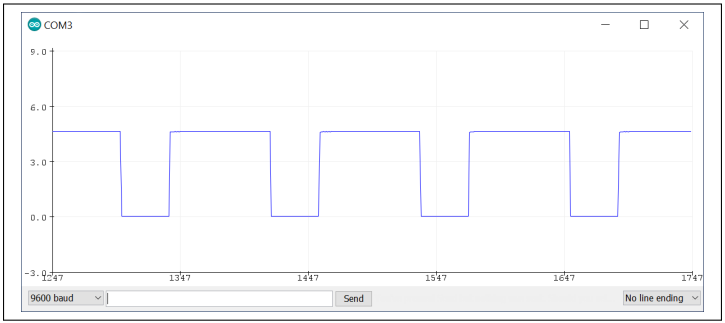
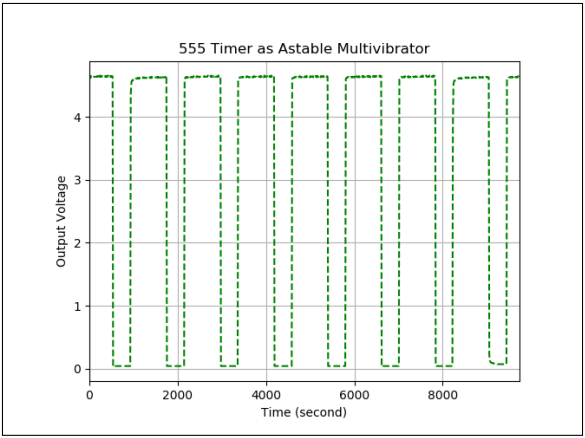


Figure: Squarewave output from 555 timer.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import os
4  import scipy as sy
5  import scipy.fftpack as syfp
6  import pylab as pyl
7
8  f1 = open('555_timer.txt','r')
9  header1=f1.readline()       # skip the header
10 data1 = np.genfromtxt(f1,delimiter='\t')
11
12 x_time=data1[:,0]/1000.0
13 y_vc=data1[:,1]
14 f1.close()
15
16 plt.plot(x_time,y_vc,"g--")
17 plt.xlabel('Time (second)')
18 plt.ylabel ('Output Voltage')
19 plt.title ('555 Timer as Astable Multivibrator')
20 plt.xlim(0,max(x_time/2.0))
21 plt.grid()
22 plt.show()
```


555 Timer as Astable Multivibrator
Output Voltage
Time (second)
0
1
2
3
4
0
2000
4000
6000
8000

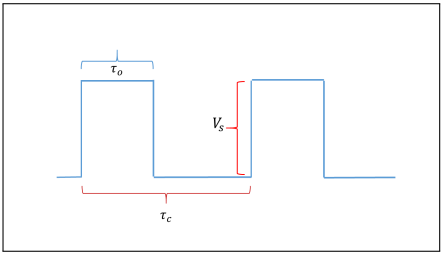
Pulse Width Modulation

is a technique for **mimicking** analog results using digital signals.

Digital control is used to create a square wave.

This repetitive *on-off* pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends ON versus the time that the signal spends OFF.

Offcourse, the response of the load which receives this signal is slower than the frequency of the pulse.



The various terms of relevance for PWM are

1. **On-Time**, the duration for which the signal is HIGH τ_O.
2. **Period**, the time taken for one complete oscillation.
3. **Duty-Cycle**, the percentage of time for which the signal remains ON, during the *period* of the signal, τ_o/τ_c.

The two main operational parameters which describes a PWM signal are the duty cycle and frequency.

By cycling a digital signal off and on at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices.

Thus the output voltage is a function of the duty cycle, for a 100% duty cycle, the output, would be 5 volt, whereas a 50% duty cycle would result in an output of 2.5 volt.

PWM using Arduino

The output of a PWM channel is available on digital I/O pins 3, 5, 6, 9, 10 and 11.

These pins can be identified as they have the symbol $\sim$ before the corresponding pin numbers.

When used, it appears as an *effective* voltage of

$$V_{eff} = V_s \frac{\tau_0}{\tau_c}$$

analogWrite(PWM_pin_no, level)

analogWrite (PWM_pin_no, level) function can be used to PWM , where the parameter level is an 8 bit integer, which can be set to obtain the desired duty cycle

We know that since the parameter is a 8 bit integer, it's maximum value is 255.

We can use the scaling of 5 $V \longrightarrow 255$, hence the desired voltage could be set as $\frac{V_o}{5} \times 255$, where V_o is the desired effective analog voltage.

```
int led = 9;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
// declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
// set the brightness of pin 9:
analogWrite(led, brightness);
// change the brightness for next time through the loop:
brightness = brightness + fadeAmount;

// reverse the direction of the fading at the ends of the fade:
if (brightness <= 0 || brightness >= 255) {
 fadeAmount = -fadeAmount;
 }
// wait for 30 milliseconds to see the dimming effect
delay(30);
}
```

Precautions

Before commencing any experiment, run the Blink code to ensure the board and the communications with the host computer are well established.

The connectors are a major source of faults.

Before commencing any experiment please ensure the continuity of the wires.

An improper connector would give us wrong voltages (or open connections).

Thanks